Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 303 (2003) 135–156

Theoretical
Computer Science

www.elsevier.com/locate/tcs

Finding approximate repetitions under Hamming distance

Roman Kolpakov[b,1], Gregory Kucherov[a]

[a] *Loria/INRIA-Lorraine, 615, rue du Jardin Botanique, B.P. 101, 54602 Villers-lès-Nancy, France*
[b] *French–Russian Institute for Informatics and Applied Mathematics, Moscow University, Moscow 119899, Russia*

Abstract

The problem of computing periodicities with K possible mismatches is studied. Two main definitions are considered, and for both of them an $O(nK \log K + S)$ algorithm is proposed (n the word length and S the size of the output). This improves, in particular, the bound obtained by G. Landan and J. Schmidt in 1993 (Proceedings of the Fourth Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, Vol. 684, Springer, Berlin, Padova, Italy, pp. 120–133). Finally, other possible definitions are briefly analyzed.

1. Introduction

Repetitions (periodicities) play a central role in word combinatorics [18,4]. On other hand, repetitions are important from the application perspective. As an example, their properties allow to speed up pattern matching algorithms [9,8,5].

The problem of efficiently identifying repetitions in a given word is one of the classical pattern matching problems [6,24]. A *tandem repeat* or a *square* is a pair of consecutive occurrences of a subword in a word. For example, *baba* is a tandem repeat in word *cbacbabacba*. Since the beginning of 80s [7], it is known that checking whether a word contains *no* tandem repeat (or is *square-free*) can be done in time $O(n)$. If one wants to find *all* tandem repeats, their number comes into consideration. Word a^n contains $O(n^2)$ tandem repeats. If we restrict ourselves to *primitive squares* (i.e. subwords uu where u is not itself a repetition v^k for $k \geqslant 2$), then a word may contain

[1] This work has been done during 1 year stay at Loria during 2000–2001.
E-mail addresses: roman@vertex.inria.msu.ru (R. Kolpakov), kucherov@loria.fr (G. Kucherov).

0304-3975/03/$ - see front matter
PII: S0304-3975(02)00448-6

$O(n \log n)$ of them and this bound is tight. All primitive squares can be found in time $O(n+S)$ where S is their number [14,25,12], hence in time $O(n \log n)$ in the worst case.

In [13,12], we studied *maximal repetitions* (see also [19,20]). Those can be viewed as maximal *runs* of squares [11,25], i.e. series of squares of equal length with contiguous start positions. For example, *bcbacacacaab* contains a maximal repetition *acacaca* which is a succession of four squares: *acac*, *caca*, *acac*, *caca*. Thus, the set of maximal repetitions can be regarded as an encoding of all tandem repeats in the string. We showed [13] that this encoding is more compact in the worst case, as there are only $O(n)$ maximal repetitions in words of length n. Moreover, all of them can be found in time $O(n)$ [12].

Recently, searching for repetitions in a string received a new motivation, due to the biosequence analysis [10]. Successive occurrences of a fragment often bear an important information in DNA sequences and their presence is characteristic for many genomic structures (such as telomer regions for example). From a practical viewpoint, satellites and alu-repeats are involved in chromosome analysis and genotyping, and thus are of major interest to genomic researchers. Thus, different biological studies based on the analysis of tandem repeats have been performed (see [27] as an example), and even databases of tandem repeats in certain species have been compiled [17].

The major difficulty in finding biologically relevant repetitions in genomic sequences is a certain variation that must be admitted between the copies of the repeated subword. In other words, biologists are interested in *approximate repetitions* and not necessarily in exact repetitions only. Finding approximate repetitions is the subject of this paper.

The simplest notion of approximate repetition is an *approximate tandem repeat* which is a subword uv where u and v are within a given distance k and the distance measure could be one of those usually used in biological applications, such as Hamming distance or edit distance. The problem of finding approximate tandem repeats for both these distances has been studied by Landau and Schmidt [15]. They showed that in case of the Hamming distance (respectively, edit distance), all approximate tandem repeats can be found in time $O(nK \log(n/K)+S)$ (respectively, $O(nK \log K \log n+S)$), where S is the number of repeats found. Several other approaches to finding approximate tandem repeats in DNA sequences have been proposed in the bioinformatics community—some of them use a statistical framework [1,2], some require to specify the size of repeated motif [3,21], some use a very general framework and have to make some heuristic filtering steps to avoid exponential blow-up [23].

This paper deals with finding approximate repetitions using exact combinatorial methods of string matching. We focus on the Hamming distance case when the variability between repeated copies can be only letter replacements. An important motivation is to define structures encoding families of approximate tandem repeats, analogous to maximal repetitions in the exact case. In Section 2, we define two fundamental structures which we call *globally-defined approximate repetitions* and *runs of approximate tandem repeats*. In Section 3, we show that all globally-defined approximate repetitions can be found in time $O(nK \log K + S)$, where S is their number. In Section 4 we show that the same bound holds for runs of approximate tandem repeats: all of them can be found in time $O(nK \log K + R)$, where R is their number. The latter result implies, in particular, that all approximate tandem repeats can be found in time $O(nK \log K + T)$

(T their number), improving the $O(nK\log(n/K)+T)$ bound of [15] for the most interesting case of small K. Finally, in Section 5 we introduce two other possible notions of approximate repetitions and give a brief analysis of their properties.

2. K-mismatch globally-defined repetitions and runs of K-mismatch tandem repeats

Quoting [1], *one difficulty in dealing with* (approximate) *tandem repeats is accurately defining them*. Intuitively, approximate repetitions are specified by authorizing some number of "errors" between repeated copies. Usually (in genomic applications at least), two types of errors are considered—letter replacements and *indels* (letter insertions/deletions). To each set of allowed errors is associated a *transformation distance* between two words, defined as the smallest number of errors that should be made to transform one word to another. If replacements only are allowed, this yields the classic *Hamming distance*, defined as the number of mismatches between the two words; if both replacements and indels are permitted we get the *edit distance* (known also as *Levenstein distance*).

In this paper, we consider the case of Hamming distance, i.e. the only allowed type of "errors" will be letter replacements. However, even in this simpler case, different notions of approximate repetitions can be thought of. Here we introduce two of them, other definitions will be discussed in Section 5.

We start by recalling briefly some facts about exact repetitions. The period of a word $w[1:n]$ is the minimal natural number p such that $w[i]=w[i+p]$ for all $1\leqslant i, i+p\leqslant n$. The ratio n/p is called the *exponent* of w. A *repetition* is any word with the exponent greater or equal to 2 [13]. A *tandem repeat*, or a *square*, is a word which is a catenation of another word with itself. Equivalently, a tandem repeat is a repetition, the exponent of which is an even natural number. In the case when the exponent is equal to 2, the tandem repeat (square) is called *primitive*. The following proposition is well known (see [18]).

Proposition 2.1. *A word $r[1:n]$ is a repetition of period $p\leqslant n/2$ if and only if one of the following conditions holds*:

(i) $r[1..n-p]=r[p+1..n]$, *and p is the minimal number with this property*,

(ii) *any subword of r of length $2p$ is a tandem repeat, and p is the minimal number with this property*.

When considering repetitions as subwords of a bigger word, the notion of maximality turns out to be very useful: a repetition is *maximal* iff it cannot be extended (by one letter) to the right or left without changing the period. Formally, given a word $w[1:n]$ and a subword $w[i..j]$ which is a repetition of exponent $e\geqslant 2$, this repetition is called *maximal* if the period of both $w[i..j+1]$ (provided that $j<n$) and $w[i-1..j]$ (provided that $i>1$) is strictly larger than e. For example, *acaabaababc* contains repetition (tandem repeat) *aabaab* which is not maximal, as the *a* which follows it respects the periodicity. On the other hand, *aabaaba* occurs as a maximal repetition. Maximal repetitions were studied in [20,13,12,25].

We now turn to defining approximate repetitions. Similar to the exact case, the basic notion here is the approximate tandem repeat. Assume $h(\cdot,\cdot)$ is the Hamming distance between two words of equal length, that is $h(w_1,w_2)$ is the number of mismatches (letter differences at corresponding positions) between w_1 and w_2. For example, $h(baaacb,bcabcb)=2$.

Definition 2.2. A word $\alpha=\alpha'\alpha''$, such that $|\alpha'|=|\alpha''|$, is called a *K-mismatch tandem repeat* iff $h(\alpha',\alpha'')\leqslant K$. Reusing the terminology of the exact case [12], we call $p=|\alpha'|=|\alpha''|$ the *period* of α, and words α', α'' the *left* and *right root* of α, respectively.

We now want to define a more global structure which would be able to capture "long approximate periodicities", generalizing repetitions of arbitrary exponent in the exact case. As opposed to the exact case, Conditions (i)–(ii) of Proposition 2.1 generalize to different notions of approximate repetition. Condition (i) gives rise to the strongest of them.

Definition 2.3. A word $r[1:n]$ is called a K-mismatch *globally-defined repetition* of period p, $p\leqslant n/2$, iff $h(r[1..n-p],r[p+1..n])\leqslant K$.

Equivalently, $r[1:n]$ is a K-mismatch globally-defined repetition of period p, if the number of i such that $r[i]\neq r[i+p]$ is at most K. For example, *abaa abba cbba cb* is a 2-mismatch globally-defined repetition of period 4. *abc abc abc abd abd abd abd abd* is a 1-mismatch globally-defined repetition of period 3 but *abc abc abc abb abc abc abc abb* is not.

Another point of view, expressed by Condition (ii) of Proposition 2.1, considers a repetition as an encoding of squares it contains [11,25]. Projecting this to the approximate case, we come up with the notion of *run of approximate tandem repeats*:

Definition 2.4. A word $r[1:n]$ is called a *run* of K-mismatch tandem repeats of period p, $p\leqslant n/2$, iff for every $i\in[1..n-2p+1]$, the subword $\alpha=r[i..i+2p-1]=r[i..i+p-1]r[i+p..i+2p-1]$ is a K-mismatch tandem repeat of period p.

Similarly to the exact case, when we are looking for approximate repetitions occurring in a word, it is natural to consider *maximal* approximate repetitions. These are repetitions extended to the right and left as much as possible provided that the corresponding definition is still verified. Note that the notion of maximality applies to both definitions of approximate repetition considered above: in both cases we can extend a repetition to the right/left as long as the obtained subword remains a repetition according to the considered definition. Throughout this paper we will be *always* interested in maximal repetitions, without mentioning it explicitly. Note that for both notions of approximate repetitions defined above, the maximality requirement implies that if $w[i:j]$ is a repetition of period p in $w[1:n]$, then $w[j+1]\neq w[j+1-p]$ (provided $j<n$)

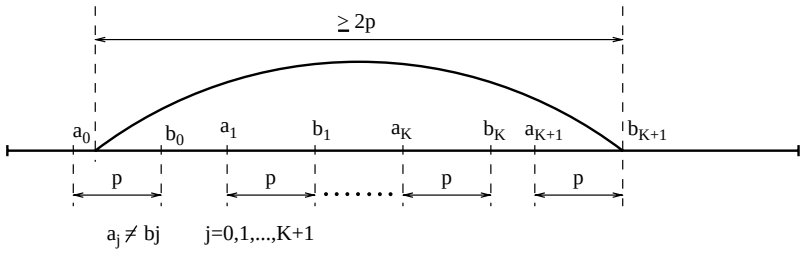

Fig. 1. Maximal K-mismatch globally-defined repetition.

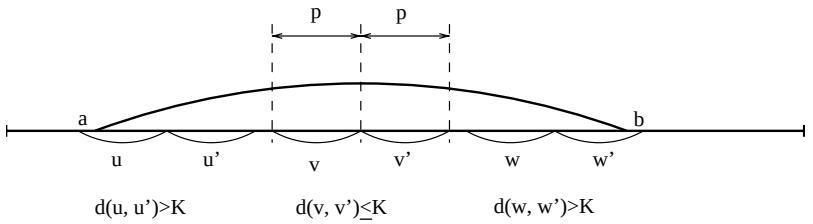

Fig. 2. Maximal run of K-mismatch tandem repeats.

and $w[i-1] \neq w[i-1+p]$ (provided $i>1$)[2]. Furthermore, if $w[i:j]$ is a maximal globally-defined repetition, it contains *exactly* K mismatches $w[l] \neq w[l+p]$, $i \leqslant l, l+p \leqslant j$, unless the whole word w contains less than K mismatches (to simplify the presentation, we always exclude this latter case from consideration).

Fig. 2 illustrates the definition of (maximal) run of K-mismatch tandem repeats, and Fig. 1 that of (maximal) K-mismatch globally-defined repetitions.

Example 2.5. The following Fibonacci word contains three runs of 3-mismatch tandem repeats of period 6. They are shown in regular font, in positions aligned with their occurrences. Two of them are identical, and contain each four 3-mismatch globally-defined repetitions, shown in italic for the first run only. The third run is a 3-mismatch globally-defined repetition in itself.

```
010010 100100 101001 010010 010100 1001

 10010 100100 101001
 10010 100100 10
  0010 100100 101
    10 100100 10100
     0 100100 101001
                  1001 010010 010100 1
                            10 010100 1001
```

[2] For one type of repetitions defined in Section 5 this implication will not hold, and we will add this condition explicitly.

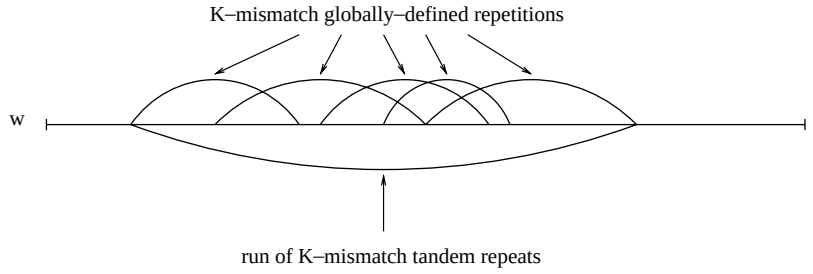

Fig. 3. Extension relation.

In general, each K-mismatch globally-defined repetition is a subword of a run of K-mismatch tandem repeats. On the other hand, a run of tandem repeats in a word is the union of all globally-defined repetitions it contains. We say then that the notion of run of K-mismatch tandem repeats *extends* that of K-mismatch globally-defined repetition (see Fig. 3).

However, a run of tandem repeats may contain as many as a linear numbers of globally-defined repetitions with the same period. For example, the word $(000\,100)^n$ of length $6n$ is a run of 1-mismatch tandem repeats of period 3, which contains $(2n-1)$ 1-mismatch globally-defined repetitions. Below is another example.

Example 2.6. The following run of 1-mismatch tandem repeats of period 4 contains eight 1-mismatch globally-defined repetitions, shown below in positions aligned with their occurrences.

```
0000 1000 1100 1110 1111 1111 0111 0011 0001 0000
0000 1000 1
 000 1000 1100 11
       00 1100 1110 111
             0 1110 1111 1111
                    1111 1111 0111 0
                           111 0111 0011 00
                                 11 0011 0001 000
                                       1 0001 0000
```

It is easily seen that the run can be iterated and therefore this gives another example of a family of runs containing a linear number of globally-defined repetitions.

In general, the following observation holds.

Lemma 2.7. *Let $w[1:n]$ be a run of K-mismatch tandem repeats of period p and let s be the number of mismatches $w[i] \neq w[i+p]$, $1 \leqslant i, i+p \leqslant n$ (equivalently, $s = h(w[1..n-p], w[p+1..n])$). Then w contains $s-K+1$ globally-defined repetitions of period p.*

Note that both notions can be criticized as for their relevance to practical situations. An obvious property of runs, as shown by Example 2.6, is that the repeated pattern can change completely along the run regardless the value of K. For example, *aaa aba abb abb bbb* is a run of 1-mismatch tandem repeats of period 3, although three-letter patterns *aaa* and *bbb* have nothing in common. On the other hand, globally-defined repetitions put a global limit on the number of mismatches and therefore may not capture some repetitions that one would possibly like to consider as such, in particular repetitions of big exponent where the total number of mismatches can exceed K while the relative number of mismatches remains low. However, these two structures are of primary importance as they provide, respectively, the weakest and strongest notions of repetitions with K mismatches, and therefore "embrace" all practically relevant repetitions. In what follows we propose efficient algorithms to find both those types of repetitions.

3. Finding K-mismatch globally-defined repetitions

In this section we describe how to find, in a given word w, all maximal K-mismatch globally-defined repetitions occurring in w (K is a given constant). Our algorithm extends, on the one hand, the one for exact maximal repetitions [20,12] and on the other hand, generalizes the one of [15] (see also [10]) by using a special factorization of the word to speed-up the algorithm.

To proceed, we need more definitions. Consider a globally-defined repetition $r = w[i..j]$ of period p in a word $w[1:n]$. $w[i..i+p-1]$ is called the *left root* of r and $w[j-p+1..j]$ its *right root*. r is said to *contain* the character $w[l]$ iff $i \leqslant l \leqslant j$, and is said to *touch* $w[l]$ iff r contains $w[l]$, or contains one of characters $w[l-1]$, $w[l+1]$.

We assume we fixed a minimal bound p_0 for the period of repetitions we are looking for. For example, p_0 can be taken to be $K+1$ having in mind that if a period $p \leqslant K$ is allowed, a tandem repeat of length $2p$ with no common characters between the left and the right root would fall under the definition. This is a purely pragmatic assumption which does not affect the method nor the complexity bounds.

Our first basic technique is described by the following auxiliary problem: Given a word $w[1:n]$ and a distinguished character $w[l]$, $l \in [2..n-1]$, we wish to find all K-mismatch globally-defined repetitions in w which touch $w[l]$. We distinguish two disjoint classes of repetitions according to whether their right root starts to the left or to the right of $w[l]$. We concentrate on repetitions of the first class, those of the second class are found similarly.

For each $p \in [p_0..l-1]$, and for all $k \in [0..K]$, we compute the following functions:

$$LP_k(p) = \max\{j \mid h(w[l-p..l-p+j-1], w[l..l+j-1]) \leqslant k\}, \tag{1}$$

$$LS_k(p) = \max\{j \mid h(w[l-p-j..l-p-1], w[l-j..l-1]) \leqslant k\}. \tag{2}$$

Informally, $LP_k(p)$ is the length of the longest subword in w starting at position $l-p$ and equal, within k mismatches, to the subword of the same length starting at l. Similarly, $LS_k(p)$ is the length of the longest subword ending at position $l-p-1$ and equal, within k mismatches, to the subword ending at position $l-1$. These functions are variants of *longest common extension functions* [15,10] and can be computed in time $O(nK)$ using suffix trees combined with the lowest common ancestor computation in a tree. We refer to [10] for a detailed description of the method.

Consider now a K-mismatch globally-defined repetition r of period p which has its right root starting to the left of $w[l]$. Note that character $w[l-p]$ is contained in r, and that r is uniquely defined by the number of mismatches $w[i-p] \neq w[i]$, $i \geqslant l$, contained in r. Let k be the number of those mismatches. Then

$$LP_k(p) + LS_{K-k}(p) \geqslant p. \tag{3}$$

Conversely, (3) can be used to detect a repetition. The following theorem holds (see [15,10]), which is a generalization of the corresponding result of [19,20] for the exact case.

Theorem 3.1. *Let $w[1:n]$ be a word and $w[l]$, $1<l<n$, a distinguished character. There exists a K-mismatch globally-defined repetition of period p which touches $w[l]$, and has its right period starting to the left of $w[l]$, iff for some $k \in [0..K]$,*

$$LP_k(p) \leqslant p, \tag{4}$$

and inequation (3) holds. In this case, this repetition starts at position $l-p-LS_{K-k}(p)$ and ends at position $l+LP_k(p)-1$.

Inequation (4) ensures that the right root starts to the left of $w[l]$.

Theorem 3.1 provides an $O(nK)$ algorithm for finding all considered globally-defined repetitions: compute longest extension functions (1) (2) (this takes time $O(nK)$) and then check inequations (3), (4) for all $p \in [p_0..l-1]$ and all $k \in [0..K]$ (this takes time $O(nK)$ too). Each time the inequations are verified, a new repetition is identified.

To find the repetitions with the right root starting to the right of $w[l]$, functions LP and LS have to be defined symmetrically: $LP_k(p)$ should be defined as the biggest j such that $h(w[l+p+1..l+p+j], w[l+1..l+j]) \leqslant k$, and $LS_k(p)$ as the biggest j such that $h(w[l+p-j+1..l+p], w[l-j+1..l]) \leqslant k$. Then inequation (3) indicates the presence of a repetition. Similarly, those repetitions can be found in time $O(nK)$.

The algorithm solving the auxiliary problem described above will be referred to as Algorithm 1. Its pseudo-code is shown below.

The second important tool is Lempel–Ziv factorization used in the well-known compression method. Let w be a word and assume that the last symbol of w does not

occur elsewhere. In this paper, we need two variants of the Lempel–Ziv factorization, that we call *with copy overlap* and *without copy overlap*[3].

Algorithm 1 Computing all K-mismatch globally-defined repetitions in w touching a distinguished character

Input: word $w[1:n]$, position l, $1<l<n$
Output: all K-mismatch globally-defined repetitions in w which touch $w[l]$
{Find those repetitions which have their right root starting to the left of $w[l]$}
1: for all $p\in[p_0..l-1]$, $k\in[0..K]$, compute longest common extension functions $LP_k(p)$, $LS_k(p)$ defined as in (1), (2)
2: **for** $p=p_0$ to $\min\{n-l+1, n/2\}$ **do**
3: **for** $k=0$ to K **do**
4: **if** $LP_k(p)+LS_{K-k}(p)\geqslant p$ and $LP_k(p)\leqslant p$ **then**
5: output a K-mismatch globally-defined repetition starting at position $l-p-LS_{K-k}(p)$ and ending at position $l+LP_k(p)-1$
{Similarly, find those repetitions which have their right root starting to the right of $w[l]$}

Definition 3.2. The Lempel–Ziv factorization of w with copy overlap (respectively without copy overlap) is the factorization $w=f_1f_2\dots f_m$, where f_i's are defined inductively as follows:

- $f_1=w$ [1],
- for $i\geqslant 2$, f_i is the shortest word occurring in w immediately after $f_1f_2\dots f_{i-1}$ which does not occur in $f_1f_2\dots f_i$ other than in suffix (respectively, does not occur in $f_1f_2\dots f_{i-1}$).

As an example, the Lempel–Ziv factorization with copy overlap of the word *aabbababababbbc* is *a|ab|ba|bababb|bc*; the factorization without copy overlap is *a|ab|ba|bab|abbb|c*. Both variants of Lempel–Ziv factorization can be computed in linear time [22,10]. If $w=f_1f_2\dots f_m$ is the Lempel–Ziv factorization, we call f_i's *Lempel–Ziv factors* or simply *factors* of w. The last character of f_i will be called the *head* of f_i.

We are now ready to describe our algorithm for finding all K-mismatch globally-defined repetitions. Consider the Lempel–Ziv factorization *with copy overlap* of w. The algorithm consists of three stages. The **first stage** is based on the following two lemmas.

Lemma 3.3. *The right root of a K-mismatch globally-defined repetition cannot contain as subword $K+1$ consecutive Lempel–Ziv factors.*

[3] The s-factorization used in [20,12] is a minor modification of the Lempel–Ziv factorization with copy overlap. The difference is that the s-factorization considers the longest factor occurring earlier, while the Lempel–Ziv factorization considers the shortest factor which does not occur earlier (see [10] for a related discussion). In this paper, we use the Lempel–Ziv factorization which better suits our purposes.

Proof. Each factor contained in the right root contains a character mismatching the one located one period to the left. Indeed, if it does not contain a mismatch, it has an exact copy occurring earlier, which contradicts the definition of factorization. As the right root contains at most K mismatches, it cannot contain $K+1$ or more factors.

We divide w into consecutive *blocks* of $K+2$ Lempel–Ziv factors. Let $w=B_1 \dots B_{m'}$ be the partition of w into such blocks. The last character of B_i will be called the *head character* of this block. At the first stage, we find, for each block B_i, those repetitions which touch the head character of B_i but do not touch that of B_{i+1}. First, concentrate on those of such repetitions with the right root starting before the head character of B_i.

Lemma 3.4. *Assume a K-mismatch globally-defined repetition r touches the head character of B_i but not that of B_{i+1}.* Then $|r| < 2|B_i B_{i+1}|$.

Proof. Lemma 3.3 implies that the right root of r cannot start before the first character of B_i. Therefore, the period of r is bounded by $|B_i B_{i+1}|$. On the other hand, by the argument of the proof of Lemma 3.3, r cannot extend by more than a period to the left of B_i. Therefore, the total length of r is bounded by $2|B_i B_{i+1}|$. □

Lemma 3.4 allows us to apply Algorithm 1: Consider word $w_i = vB_iB_{i+1}$, where v is the suffix of $B_1 \dots B_{i-1}$ of length $|B_i B_{i+1}|$. Then find, using Algorithm 1, all repetitions in w_i touching the head character of B_i and discard those which touch the head character of B_{i+1}. The resulting complexity is $O(K(|B_i| + |B_{i+1}|))$.

After processing all blocks, we find all repetitions touching block head characters. Observe that repetitions resulting from processing different blocks are distinct. Summing up over all blocks, the resulting complexity of the first stage is $O(nK)$. The repetitions which remain to be found are those which lie entirely within a block—this is done at the next two stages.

At the **second stage** we find all repetitions inside each block B_i which touch factor heads other than the block head (=head character of the last factor of the block). For each B_i, we proceed by simple binary division approach:

(i) divide current block of factors $B = f_i f_{i+1} \dots f_{i+m}$ into two sub-blocks $B' = f_i \dots f_{\lfloor m/2 \rfloor}$ and $B'' = f_{\lfloor m/2 \rfloor + 1} \dots f_{i+m}$,

(ii) using Algorithm 1, find the repetitions in B which touch the head character of $f_{\lfloor m/2 \rfloor}$, but discard those which touch the head character of f_{i+m} or contain the first character of f_i,

(iii) process recursively B' and B''.

The above algorithm has $\lceil \log K \rceil$ levels of recursion, and since at each step the word is split into disjoint sub-blocks, the whole complexity of the second stage is $O(nK \log K)$.

Finally, at the **third stage**, it remains to find the repetitions which occur entirely inside each Lempel–Ziv factor, namely which do not contain its first character and do not touch its head character. By definition of the factorization with copy overlap (Definition 3.2), each factor without its head character has another (possibly overlapping)

occurrence to the left. Therefore, each of these repetitions has another occurrence to the left too. Using this observation, these repetitions can be found using the same technique as the one of [12]: When constructing the Lempel–Ziv factorization we keep for each factor wa a pointer to a copy of w to the left. Then processing factors from left to right, recover repetitions inside the factor from its pointed copy. We refer to [12] for algorithmic details. The complexity of this stage is $O(n+S)$, where S is the number of repetitions found.

The following theorem summarizes this section.

Theorem 3.5. *All K-mismatch globally-defined repetitions can be found in time $O(nK\log K+S)$ where n is the word length and S is the number of repetitions found.*

The algorithm of finding all K-mismatch globally-defined repetitions, referred to as Algorithm 2, is given below.

4. Finding runs of K-mismatch tandem repeats

In this section we describe an algorithm for finding all runs of K-mismatch tandem repeats in a word.

The general structure of the algorithm is the same as for globally-defined repetitions (Algorithm 2)—it has the three stages playing similar roles. At the first and second stages, the key difference is the type of objects we are looking for: instead of computing globally-defined repetitions we now compute *subruns* of K-mismatch tandem repeats. Formally, a subrun is a run of K-mismatch tandem repeats, which is not necessarily maximal. At each point of the first and second stage when we search for repetitions touching some head character $w[l]$, we now compute subruns of those K-mismatch tandem repeats which touch $w[l]$. This can be seen as outputting by Algorithm 1 only that part of the globally-defined repetition which falls to the interval $l-2p..l+2p$. The modified Algorithm 1, referred to as Algorithm 3, is given below.

Algorithm 2 Computing all K-mismatch globally-defined repetitions in w

---

Input: word $w[1:n]$

Output: all K-mismatch globally-defined repetitions in w

1: Compute the Lempel–Ziv factorization with copy overlap $w=f_1\dots f_m$
2: Partition the factorization into blocks of $K+2$ consecutive factors; let $w=B_1\dots B_{m'}$ be the decomposition of w into such blocks
{**first stage**}
3: **for** each block B_i **do**
4: find, using Lemma 3.4 and Algorithm 1, globally-defined repetitions which touch the head character of B_i but not that of B_{i+1}
{**second stage**}

5: **for** each block B_i **do**
6: starting from B_i apply the following recursive procedure
7: divide the current block $B = f_i f_{i+1} \dots f_{i+m}$ into two sub-blocks $B' = f_i \dots f_{\lfloor m/2 \rfloor}$ and $B'' = f_{\lfloor m/2 \rfloor + 1} \dots f_{i+m}$
8: find, using Algorithm 1, globally-defined repetitions in B which touch the last character of $f_{\lfloor m/2 \rfloor}$, but discard those which touch the head character of f_{i+m} or contain the first character of f_i
9: process recursively B' and B''
{**third stage**}
10: **for** each Lempel–Ziv factor f_j **do**
11: retrieve all globally-defined repetitions in f_j which do not contain its first character and don't touch its last character, from its left copy (see [12] for details)

A major additional difficulty in computing runs is *assembling* subruns into runs. To perform the assembling, we need to store subruns in an additional data structure and to carefully manage merging of subruns into bigger runs. We have to ensure that the number of subruns we come up with and the work spent on processing them do not increase the resulting complexity bound.

The assembling occurs already at the level of Algorithm 3, as subruns found for different values of k (**for**-loop at line 3) may overlap or immediately follow each other, in which case we join them into a bigger subrun (lines 6–7). Similarly, subruns of tandem repeats with the right root starting to the right of $w[l]$ (case non-shown in Algorithm 3) may have to be joined with subruns found by instructions 1–7 of Algorithm 3). We leave out the details of how this is done.

Algorithm 3 Computing subruns of K-mismatch tandem repeats in w touching a distinguished character

Input: word $w[1:n]$, position l, $1 < l < n$
Output: all subruns of K-mismatch tandem repeats in w which touch $w[l]$
{Find those tandem repeats which have their right root starting to the left of $w[l]$}
1: for all $p \in [p_0..l-1]$, $k \in [0..K]$, compute longest common extension functions $LP_k(p)$, $LS_k(p)$ defined as in (1), (2)
2: **for** $p = p_0$ to $\min\{n - l + 1, n/2\}$ **do**
3: **for** $k = 0$ to K **do**
4: **if** $LP_k(p) + LS_{K-k}(p) \geqslant p$ and $LP_k(p) \leqslant p$ **then**
5: create a subrun of K-mismatch tandem repeats ending at positions $start(p,k) = \max\{l + p - LS_{K-k}(p) - 1, l - 1\}$ through $end(p,k) = \min\{l + LP_k(p) - 1, l + p - 1\}$

6: **if** $k>0$ and $end(p,k)\leqslant end(p,k-1)+1$ **then**
7: merge this subrun with the subrun computed for the previous value of k
{Similarly, find those tandem repeats which have their right root starting to the right of $w[l]$ and thus end in the interval $l+p..l+2p$}

Below we describe the three stages of the algorithm in more details. We identify a subrun with the interval of *end positions* of the tandem repeats it contains. This is a technical, but important convention, which allows us to express the algorithm in terms of maintaining, for each period p, a set of non-intersecting intervals corresponding to subruns of period p found so far.

For the input word w, we compute the Lempel–Ziv factorization *without copy overlap* and divide it into blocks $B_1 \dots B_{m'}$, each containing $K+2$ consecutive Lempel–Ziv factors. At the **first stage**, we find subruns of all those tandem repeats which touch block head characters. For each block B_i, we find the tandem repeats which touch the head character of B_i but not that of B_{i+1}. Let l_i be the position of the head character of B_i. Then the subruns of period p found at this step belong to the interval $[l_i-1..\min\{l_i+2p, l_{i+1}-2\}]$. We call this interval the *explored interval* for $w[l_i]$ and p. For each period, subruns found at this step can be seen as non-intersecting subintervals of this explored interval. These subruns are stored into a double-linked list in increasing order of positions. (We leave it to the reader to check out that such a list can be easily computed by Algorithm 3 by making at each step a constant amount of extra work.) For $p>(l_i-l_{i-1})/2-1$, the explored interval for $w[l_{i-1}]$ has to be merged with the explored interval for $w[l_i]$, thus forming a bigger explored interval. Accordingly, the lists of subruns associated with these intervals are merged into a single list. All additional operations take constant time, and the resulting complexity of the first stage is $O(nK)$.

The **second stage** is modified in a similar way. Recall that at each call of Algorithm 3 we are searching for repetitions occurring between some factor head, say $w[l']$, another factor head $w[l'']$, and touching some factor head $w[l]$ $(l'<l<l'')$. Assuming that recursive calls are executed in a top–down manner (see the description of the second stage in Algorithm 2), no factor head between $w[l']$ and $w[l'']$ has been processed yet. In this case, the explored interval is $[\max\{l'+2p+1, l-1\}..\min\{l+2p, l''-2\}]$, and we may have to merge it either with the previous explored interval, or with the next one, or both. The complexity of the second stage stays $O(nK\log K)$.

Note at this point that the algorithm maintains, *for each period* p, a set of non-intersecting explored intervals. Each interval is associated with a sequence of successive head characters $w[l_i], w[l_{i+1}], \dots, w[l_m]$ such that $l_{j+1}-l_j\leqslant 2p+2$ for $j\in[i..m-1]$, and the interval itself is $[l_i-1..l_m+2p]$. Those subruns of tandem repeats which have been actually found within this interval, are stored in a double-linked list associated to the interval. When two explored intervals are merged into a bigger one at the first and second stages, the corresponding lists are concatenated. Furthermore, the interval is linked to $w[l_i]$ and $w[l_m]$—the first and the last head characters of this sequence. More precisely, if $w[l_i]$ is the first head character of an interval, and p is the corresponding

period, then $w[l_i]$ has a pointer to the first subrun of period p found in this interval. Similarly, if $w[l_m]$ is the last head character of an interval of period p, then $w[l_m]$ points to the last subrun of the interval. These pointers are needed, in particular, for merging explored intervals at the second stage. Note finally that storing these pointers, for both the first and the second stage, can be done within the given complexity bound. The key observation is that, for a head character $w[l]$, a pointer to the first subrun should be defined only for periods $p \leqslant (l - l')/2 - 1$, where $w[l']$ is the closest head character to the left of $w[l]$ which has been processed before. Similarly, a pointer to the last subrun is defined only for periods $p \leqslant (l'' - l)/2 - 1$, where $w[l'']$ is the closest head character to the right of $w[l]$ which has been processed before. This shows that at each moment there is only $O(n)$ pointers that need to be stored.

At the **third stage**, we have to find subruns of those tandem repeats which lie entirely inside Lempel–Ziv factors. For each period, potential occurrences of these subruns correspond precisely to the gaps between explored intervals. Thus, the third stage can be also seen as closing up, for each period, the gaps between explored intervals.

As in the previous section, the key observation here is the fact that Lempel–Ziv factors without their head character have a copy to the left (here required to be non-overlapping), and the idea is again to process w from left to right and to retrieve the subruns inside each factor from its copy. However, the situation here is different in comparison to globally-defined repetitions: we may have to "cut out" from a longer list a chain of subruns belonging to a factor copy and then to "fit" it into the gap between two explored intervals. The "cutting out" may entail splitting subruns which span over the borders of the factor copy, and "fitting into" may entail merging those subruns with subruns from the neighboring explored intervals. Below we sketch the algorithm for the third stage, which copes with these difficulties. Algorithm 4 given below provides a detailed description of the third stage.

Algorithm 4 Third stage of the algorithm of finding runs of K-mismatch tandem repeats

Input: word $w[1:n]$; lists of subruns found at the first and second stages
Output: all runs of K-mismatch tandem repeats in w

```
        {activerun(p) is maintained to be the last considered run of period p}
        {startingruns(i) is maintained to be the list of runs starting at i}
 1:     for each position i∈[1..n] do
 2:       for each run r startingruns(i) do
 3:         let p be the period of r
 4:         activerun(p):=r
 5:         if r is not the last run in its list then
 6:           let i' be the first position of the next run in the list
 7:           add the next run to startingruns(i')
 8:       for each factor copy ending at position i do
 9:         let w[j..i] be this copy and f_m the corresponding factor
10:         for each period p⩽(j − i + 1)/2 do
11:           if activerun(p) contains tandem repeats inside w[j..i] then
```

12: link/merge this subrun of tandem repeats to/with the first subrun of the explored interval associated with the head symbol of f_m
13: *currentrun*:=the predecessor of *activerun*(p) in its list
14: **while** *currentrun* contains tandem repeats inside $w[j..i]$ **do**
15: link the subrun of those tandem repeats to the previously copied subrun in f_m
16: *currentrun*:= the predecessor of *currentrun* in its list
17: link/merge the last processed subrun to/with the last subrun of the explored interval associated with the head symbol of f_{m-1}
18: **else**
19: link the last subrun of the explored interval associated with the head symbol of f_{m-1} with the first subrun of the explored interval associated with the head symbol of f_m
20: **if** *activerun*(p) is the last subrun of the explored interval associated with the head symbol of f_{m-1} **then**
21: let i' be the first position of the next run in the list
22: add the next run (if any) to *startingruns*(i')
23: close up the gap corresponding to f_m by merging intervals associated to the head symbol of f_m and the head symbol of f_{m-1}

During the computation of the Lempel–Ziv factorization, for each Lempel–Ziv factor $f_i = va$ we choose a copy of v occurring earlier and point from the end position of this copy to the head character a of f_i. It may happen that one position has to have several pointers, in which case we organize them in a list. We traverse w from left to right and maintain the rightmost run, of each period, which starts before the current position. This run is denoted *activerun*(p) in Algorithm 4. To this purpose, we also maintain the following invariant: at the moment we arrive at a position i, we know the list *startingruns*(i) of all subruns which start at this position. *startingruns* is maintained according to the following general rule: for each subrun starting at the current position, we assign the starting position of the next subrun in the list (instructions 2–7 in Algorithm 4). Of course, there may be no next subrun if the current subrun is the last one in the explored interval. In this case, the starting position of the subrun following the current subrun will be set at the moment we fill the gap after this explored interval (instructions 20–22).

When we arrive at the end position of a copy of a Lempel–Ziv factor, we need to copy into the factor all the subruns which this copy contains. Therefore, we scan *backwards* the subruns contained in the copy and copy them to the factor (instructions 11–19). Copying the subruns closes up two explored intervals into one interval, and links together two lists of subruns, possibly inserting a new list of runs in between. Copying subruns in the backward direction is important for the correction of the algorithm—this guarantees that no subruns are missed. It is also for this reason that we need the copy to be non-overlapping with the factor.

After the whole word has been traversed, no more gaps between explored intervals exist anymore. This means that for each period, we have a list of subruns with this period occurring in the word, which are actually the searched runs.

The complexity of the third stage is $O(n+S)$, where S is the number of resulting runs. To show this, we make an amortized analysis of Algorithm 4 [26]. Specifically, we show that the *total* number of iterations of each loop in Algorithm 4 is either $O(n)$ or $O(S)$. Each iteration of the **for**-loop at line 2 processes a new run starting at position i. Therefore there are $O(S)$ iterations of this loop during the whole run. Each iteration of the **for**-loop at line 8 treats a copy of a distinct Lempel–Ziv factor. Furthermore, the number of iterations of the nested **for**-loop at line 10 is the half of the length of the corresponding factor. Therefore, the total number of iterations of both **for**-loops is $O(n)$. On the other hand, the **while**-loop at line 14 iterates $O(S)$ times, as at each iteration it treats a subrun, which becomes a definite run at that point. Thus, the *overall* time spent by all internal loops is $O(n+S)$. The main **for**-loop (line 1) makes obviously $O(n)$ iteration, and this completes the proof that the whole complexity of the third stage is indeed $O(n+S)$.

Putting together the three stages, we obtain the main result of this section.

Theorem 4.1. *All runs of K-mismatch tandem repeats can be found in time $O(nK\log K+S)$ where n is the word length and S is the number of runs found.*

Once all runs have been found, we can easily output all tandem repeats. We then have the following result improving the bound obtained in [15].

Corollary 4.2. *All K-mismatch tandem repeats can be found in time $O(nK\log K+S)$ where n is the word length and S is the number of tandem repeats found.*

5. Other notions of approximate repetitions

K-mismatch globally-defined repetitions limit by K the *total* number of mismatches, and therefore provide the strongest notion of approximate repetition. On the other hand, a run of K-mismatch tandem repeats provides the weakest notion of repetition with K mismatches, as it imposes only the minimal requirement that every tandem repeat in such a repetition contains no more than K mismatches. For practical applications, such as genome analysis, it might be interesting to consider intermediate definitions with respect to the two "extreme" cases. In this section, we introduce two such types of repetitions and point out very briefly some of their properties.

One natural way to loosen the notion of a globally-defined repetition of period p is to limit by K the Hamming distance between two subwords of length p or less, located within any distance which is a multiple of p.

Definition 5.1. A word $r[1:n]$ is called a *K-mismatch uniform repetition* of period p, $p\leqslant n/2$ iff for every two subwords $r[i..i+j-1], r[i+kp..i+kp+j-1]$ of r such that k is any integer and $1\leqslant j\leqslant p$, we have $h(r[i..i+j-1], r[i+kp..i+kp+j-1])\leqslant K$.

For example, *cbca bbca ab* is a 1-mismatch uniform repetition of period 4, but *abca abca bc* is not since *ab* at position 1 is at Hamming distance 2 from *bc* at position 9. globally-defined repetitions is a weaker notion than K-mismatch globally-defined repetitions: any word from $(abcadc)^+$ is a 1-mismatch uniform repetition of period 3, whereas none of these words is a 1-mismatch globally-defined repetition.

The following technical remark concerning uniform repetitions is important. When we consider *maximal* uniform repetition in a word we have to add an additional condition: $w[i..j]$ is a maximal K-mismatch uniform repetition of period p in w if neither $w[i-1..j]$ nor $w[i..j+1]$ is a K-mismatch uniform repetition of period p and the following additional conditions hold:

$$w[i-1] \neq w[i-1+p], \quad w[j+1-p] \neq w[j+1] \tag{5}$$

In contrast to both notions of repetition considered before, inequalities (5) do not hold automatically for uniform repetitions and should be added explicitly in order to ensure, in particular, that every K-mismatch uniform repetition in a word is the union of K-mismatch globally-defined repetitions containing in it. The following example illustrates the situation.

Example 5.2. Consider the following word of length $4p$. It contains two maximal K-mismatch globally-defined repetitions of period p starting at position $p+1$ and shown below in regular font. Both of these repetitions are also maximal K-mismatch uniform repetitions. However, if we do not require inequalities (5) to hold, we obtain a series of $K-1$ "superfluous" K-mismatch uniform repetitions shown in italic.

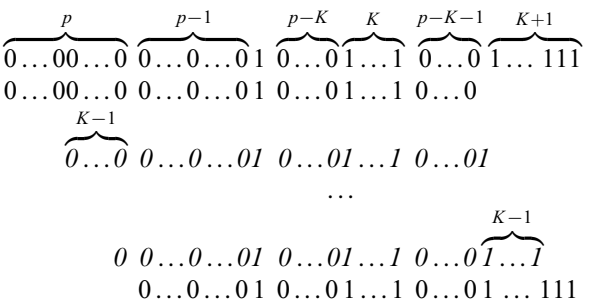

Fig. 4 gives an illustration to the definition of maximal K-mismatch uniform repetition. The relationship of uniform repetitions to globally-defined repetitions and runs are summarized in the following lemma.

Lemma 5.3. (i) *Any K-mismatch globally-defined repetition can be extended to a (possibly not unique) K-mismatch uniform repetition. Any K-mismatch uniform repetition is the union of K-mismatch globally-defined repetitions it contains.*

(ii) *Any K-mismatch uniform repetition can be extended to a unique run of K-mismatch tandem repeats. A run of K-mismatch tandem repeats is the union of K-mismatch uniform repetition it contains.*

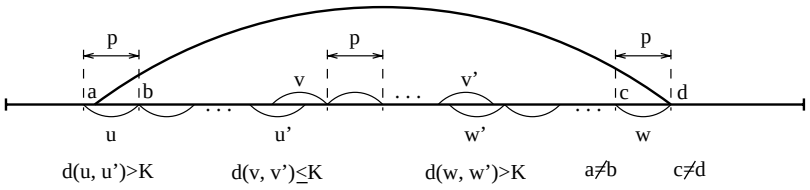

Fig. 4. K-mismatch uniform repetition.

Another possible definition of approximate repetition is obtained when one thinks of it as an "exact repetition with no more than K replacement errors per period". This viewpoint is somewhat similar to the one of [23], where a repetition is defined through a *consensus* such that each repeated motif is within a specified distance from the consensus.

Definition 5.4. Word $r[1:n]$ is a *K-mismatch consensus repetition* of period p, $p \leqslant n/2$, iff there exists an exact repetition $v[1:n]$ of period p such that for any subword $r[i..j]$ of r such that $j - i \leqslant p$, we have $h(r[i..j], v[i..j]) \leqslant K$.

Example 5.5. Consider the word from Example 2.6. The 1-mismatch consensus repetitions it contains are shown together with a possible consensus for each of them shown below in italic.

0000 1000 1100 1110 1111 1111 0111 0011 0001 0000
0000 1000 1100 11
1000 1000 1000 10
000 1000 1100 1110 111
100 1100 1100 1100 110
00 1100 1110 1111 1111
10 1110 1110 1110 1110
0 1110 1111 1111 0111 0
1 1111 1111 1111 1111 1
1111 1111 0111 0011 00
0111 0111 0111 0111 01
111 0111 0011 0001 000
011 0011 0011 0011 001
11 0011 0001 0000
01 0001 0001 0001

The notion of K-mismatch consensus repetition is illustrated on Fig. 5. Similar to uniform repetitions, consensus repetitions provide an intermediate structure between globally-defined repetitions and runs:

Lemma 5.6. *Assume K is even.*

(i) *Any K-mismatch globally-defined repetition can be extended to a (possibly not unique) $K/2$-mismatch consensus repetition. Any $K/2$-mismatch consensus*

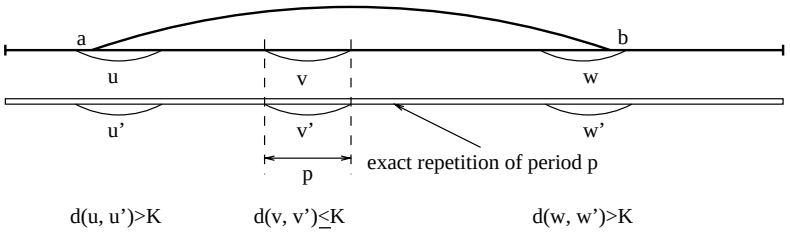

Fig. 5. K-mismatch consensus repetition.

repetition is the union of K-mismatch globally-defined repetitions it contains.

(ii) *Any $K/2$-mismatch consensus repetition can be extended to a unique run of K-mismatch tandem repeats. A run of K-mismatch tandem repeats is the union of $K/2$-mismatch consensus repetition it contains.*

Concerning the relationship between uniform and consensus repetitions, it is easily seen that a $K/2$-mismatch consensus repetition can be extended to a K-mismatch uniform repetition. Any K-mismatch uniform repetition can be, in turn, extended to a K-mismatch consensus repetition. A more subtle relationship between these two notions require an additional analysis. Designing an efficient algorithm for finding these types of repetitions remains an additional open problem.

Relations between different notions of repetition, studied in this paper, are summarized on Fig. 6. A solid arrow denotes the extension relation (see Fig. 3 and remark after Example 2.5), and a flashed arrow means just that the "source structure" can be extended to the "target structure", but not necessarily that a target structure is the union of the source structures contained in it.

6. Concluding remarks

We proposed $O(nK \log K + S)$ algorithms for finding K-mismatch globally-defined repetitions and runs of K-mismatch tandem repeats (S the output size). Note that if K is considered constant, we have $O(n+S)$ algorithms for finding each of these structures. This is an interesting result, which had been long time unknown even for the exact case [14,25,12].

In the final stage of preparation of this paper, we got known of paper [16]. In this paper, yet another notion of approximate repetitions is considered, which is weaker than globally-defined repetitions, but stronger than both uniform repetitions and consensus repetitions. The algorithm presented in [16] runs in time $O(nKE \log(n/K))$, where E is the maximal exponent of reported repetitions.

The algorithms presented in this paper has been implemented within the `mreps` software[4], specialized in the search of repetitions in DNA data. As expected, the

[4] http://www.loria.fr/mreps/.

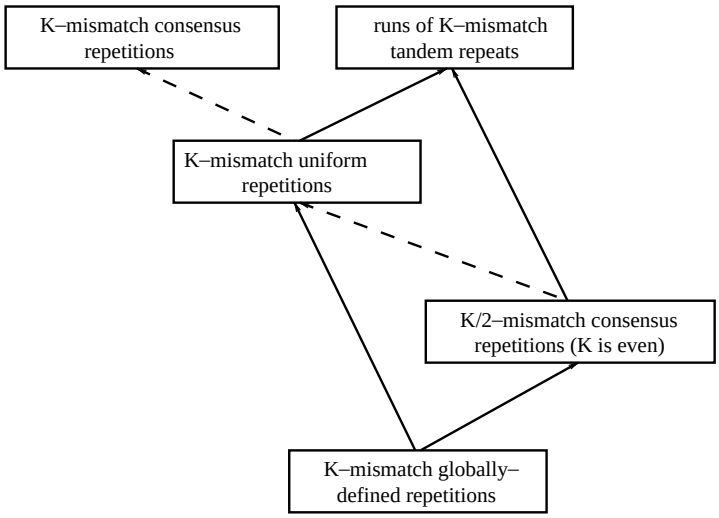

Fig. 6. Relations between different notions of repetitions.

implementation turned out to be very efficient—a sequence of several millions of letters (typical size of a bacterial genome) is processed within seconds on a regular Pentium™ computer. The experiments show that a typical run of approximate tandem repeats in a DNA sequence contains 2–3 globally-defined repetitions, and the phenomenon of error accumulation in runs does not really occur in these data. Our observation is that, in the case of DNA sequences, runs of approximate tandem repeats provide a notion which captures well biologically interesting repetitions.

Acknowledgements

We thank Mathieu Giraud, with whom we had first discussions on the subject. At early stages, this work has been supported by the REMAG project of INRIA, and by a joint project of the French–Russian A.M. Liapunov Institute of Applied Mathematics and Informatics at Moscow University. The first author has been in part supported by Russian Foundation of Fundamental Research, under grant 99-01-01175, and by Russian Federal Program “Integration”, under grant 473. We thank anonymous referees for valuable remarks.

References

[1] G. Benson, An algorithm for finding tandem repeats of unspecified pattern size, in: S. Istrail, P. Pevzner, M. Waterman (Eds.), Proc. 2nd Annu. Internat. Conf. on Computational Molecular Biology (RECOMB 98), ACM Press, New York, 1998, pp. 20–29.

[2] G. Benson, Tandem repeats finder: a program to analyze DNA sequences, Nucleic Acids Res. 27 (2) (1999) 573–580.

[3] G. Benson, M. Waterman, A method for fast database search for all k-nucleotide repeats, Nucleic Acids Res. 22 (1994) 4828–4836.

[4] C. Choffrut, J. Karhumäki, Combinatorics of words, in: G. Rozenberg, A. Salomaa (Eds.), Handbook on Formal Languages, Vol. I, Springer, Berlin, Heidelberg, New York, 1997, pp. 329–438.

[5] R. Cole, R. Hariharan, Approximate string matching: a simpler faster algorithm, in: Proc. 9th Annu. ACM-SIAM Symp. on Discrete Algorithms, San Francisco, CA, 1998, pp. 463–472.

[6] M. Crochemore, An optimal algorithm for computing the repetitions in a word, Inform. Process. Lett. 12 (1981) 244–250.

[7] M. Crochemore, Recherche linéaire d'un carré dans un mot, C. R. Acad. Sci. Paris Sér. I Math. 296 (1983) 781–784.

[8] M. Crochemore, W. Rytter, Squares, cubes, and time-space efficient string searching, Algorithmica 13 (1995) 405–425.

[9] Z. Galil, J. Seiferas, Time-space optimal string matching, J. Comput. System Sci. 26 (3) (1983) 280–294.

[10] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.

[11] C. Iliopoulos, D. Moore, W. Smyth, A characterization of the squares in a Fibonacci string, Theoret. Comput. Sci. 172 (1997) 281–291.

[12] R. Kolpakov, G. Kucherov, Finding maximal repetitions in a word in linear time, in: Proc. 1999 Symp. on Foundations of Computer Science, New York (USA), IEEE Computer Society, Silver Spring, MD, 1999, pp. 596–604.

[13] R. Kolpakov, G. Kucherov, On maximal repetitions in words, in: Proc. 12th Internat. Symp. on Fundamentals of Computation Theory, 1999, Iasi (Romania), Lecture Notes in Computer Science, Springer, Berlin, 1999, pp. 374–385.

[14] S.R. Kosaraju, Computation of squares in string, in: M. Crochemore, D. Gusfield (Eds.), Proc. 5th Annu. Symp. on Combinatorial Pattern Matching, Lecture Notes in Computer Science, Vol. 807, Springer, Berlin, 1994, pp. 146–150.

[15] G. Landau, J. Schmidt, An algorithm for approximate tandem repeats, in: A. Apostolico, M. Crochemore, Z. Galil, U. Manber (Eds.), Proc. 4th Annu. Symp. on Combinatorial Pattern Matching, Lecture Notes in Computer Science, Vol. 684, Springer, Berlin, Padova, Italy, 1993, pp. 120–133.

[16] G. Landau, J. Schmidt, D. Sokol, An algorithm for approximate tandem repeats, J. Comput. Biol. 8 (1) (2001) 1–18.

[17] P. Le Fleche, Y. Hauck, L. Onteniente, F. Prieur, A. Denoeud, V. Ramisse, P. Sylvestre, G. Benson, F. Ramisse, G. Vergnaud, A tandem repeats database for bacterial genomes: application to the genotyping of *Yersinia pestis* and *Bacillus anthracis*, BMC Microbiol. 1 (1) (2001) 2.

[18] M. Lothaire, Combinatorics on Words, in: Encyclopedia of Mathematics and its Applications, Vol. 17, Addison-Wesley, Reading, MA, 1983.

[19] M. Main, R. Lorentz, An $O(n \log n)$ algorithm for finding all repetitions in a string, J. Algorithms 5 (3) (1984) 422–432.

[20] M.G. Main, Detecting leftmost maximal periodicities, Discrete Appl. Math. 25 (1989) 145–153.

[21] E. Rivals, O. Delgrange, J.-P. Delahaye, M. Dauchet, A first step towards chromosome analysis by compression algorithms, in: N. Bourbakis (Ed.), Proc. 1st IEEE Symp. on Intelligence in Neural and Biological Systems (INBS), IEEE Computer Society, Herndon, VA, 1995.

[22] M. Rodeh, V. Pratt, S. Even, Linear algorithm for data compression via string matching, J. ACM 28 (1) (1981) 16–24.

[23] M.-F. Sagot, E. Myers, Identifying satellites in nucleic acid sequences, in: S. Istrail, P. Pevzner, M. Waterman (Eds.), Proc. 2nd Annu. Internat. Conf. on Computational Molecular Biology (RECOMB 98), ACM Press, New York, 1998, pp. 234–242.

[24] A. Slisenko, Detection of periodicities and string matching in real time, J. Soviet Math. 22 (1983) 1316–1386.

[25] J. Stoye, D. Gusfield, Linear time algorithms for finding and representing all the tandem repeats in a string, Technical Report CSE-98-4, Computer Science Department, University of California, Davis, 1998.

[26] R.E. Tarjan, Amortized computational complexity, SIAM J. Algebraic Discrete Methods 6 (2) (1985) 306–318.

[27] A. van Belkum, S. Scherer, W. van Leeuwen, D. Willemse, L. van Alphen, H. Verbrugh, Variable number of tandem repeats in clinical strains of *Haemophilus influenzae*, Infect. Immun. 65 (12) (1997) 5017–5027.